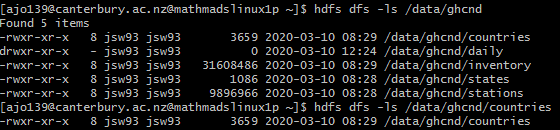
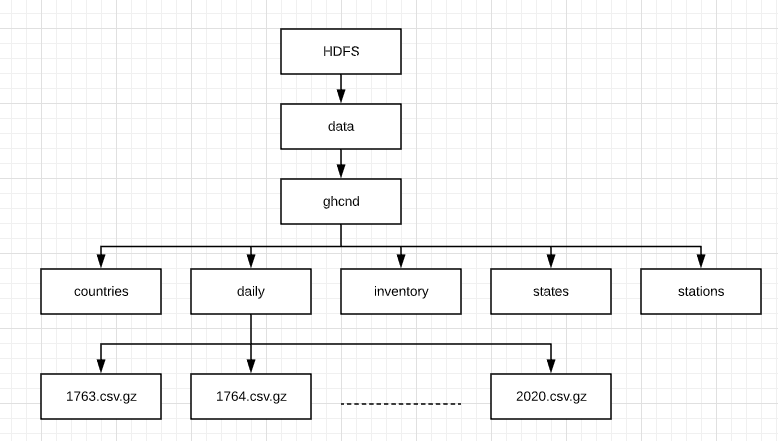
**Q1. Deﬁne the separate data sources as daily, stations, states, countries, and inventory respectively. All of the data is stored in HDFS under hdfs:///data/ghcnd and is read only. Do not copy any of the data to your home directory.**

**Use the hdfs command to explore hdfs:///data/ghcnd without actually loading any data into memory.**

1. **How is the data structured? Draw a directory tree to represent this in a sensible way.**

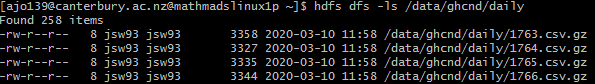
The dataset ‘ghcnd’ is located in the directory ‘data’ in hdfs and consists of 4 different files and a subdirectory ‘daily’. There are 258 files in the subdirectory ‘daily’.





1. **How many years are contained in daily, and how does the size of the data change?**

Each file in daily represents each year. Thus, we have data from 258 different years, starting from 1763 until 2020. Size of the data starts from 3358 bytes in 1763, that almost doubles in 12 years. Thereafter we can see a steady increase in the data and reached on its peak in 2010 and slightly got stabilized after that. Two months data for the year 2020 is also accessible from the directory.



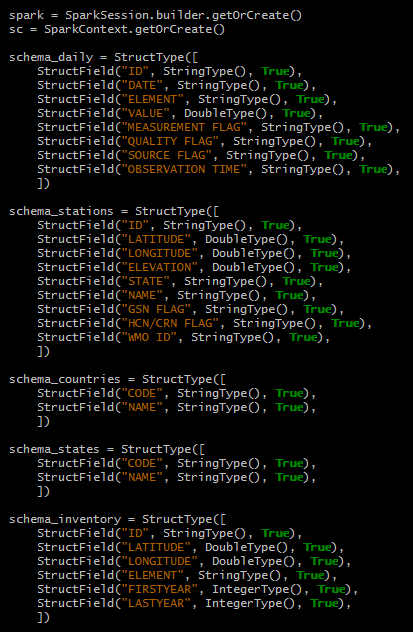
1. **What is the total size of all of the data? How much of that is daily?**

The actual size of the ghcnd data in bytes is 16680610588, and 16639100391 of those bytes is daily.



**Q2. Start pyspark shell with 2 executors, 1 core per executor, 1 GB of executor memory, and 1 GB of master memory. You will now explore each data source brieﬂy to ensure that the descriptions are accurate and that the data is as expected.**

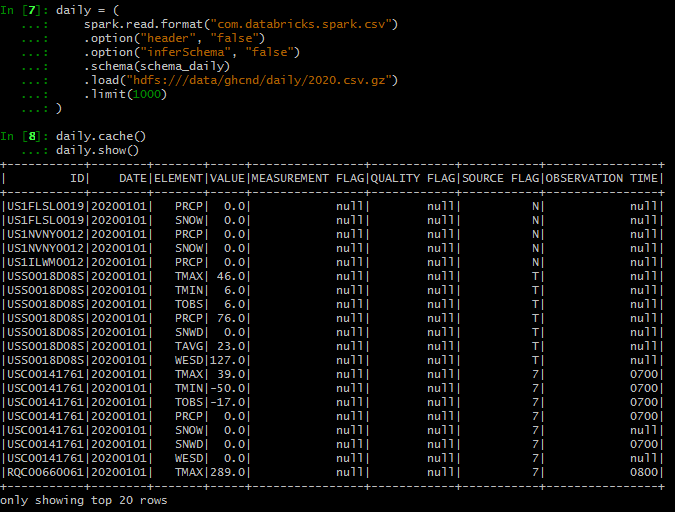
1. **Define schemas for each of daily, stations, states, countries, and inventory based on the descriptions in this assignment and the GHCN Daily README. These should use the data types defined in pyspark.sql.**



**(b) Load 1000 rows of hdfs:///data/ghcnd/daily/2020.csv.gz into Spark using the limit command immediately after the read command.**

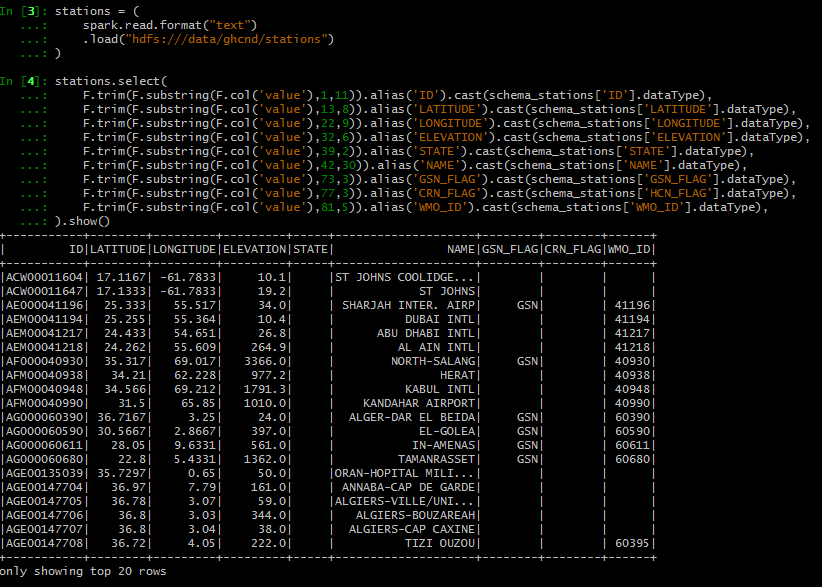
**Was the description of the data accurate? Was there anything unexpected?**

The description of the data is accurate enough. The ‘ID’ column which is expected to be a primary key is not unique.

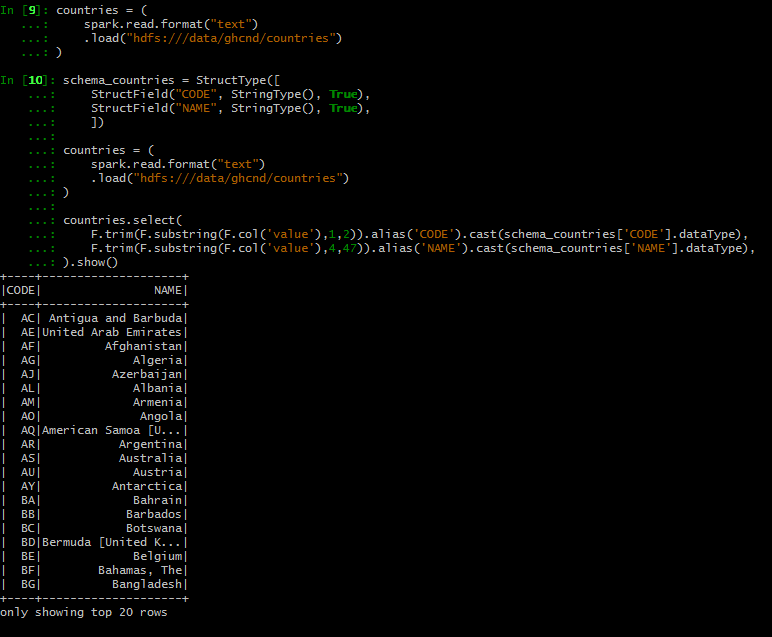


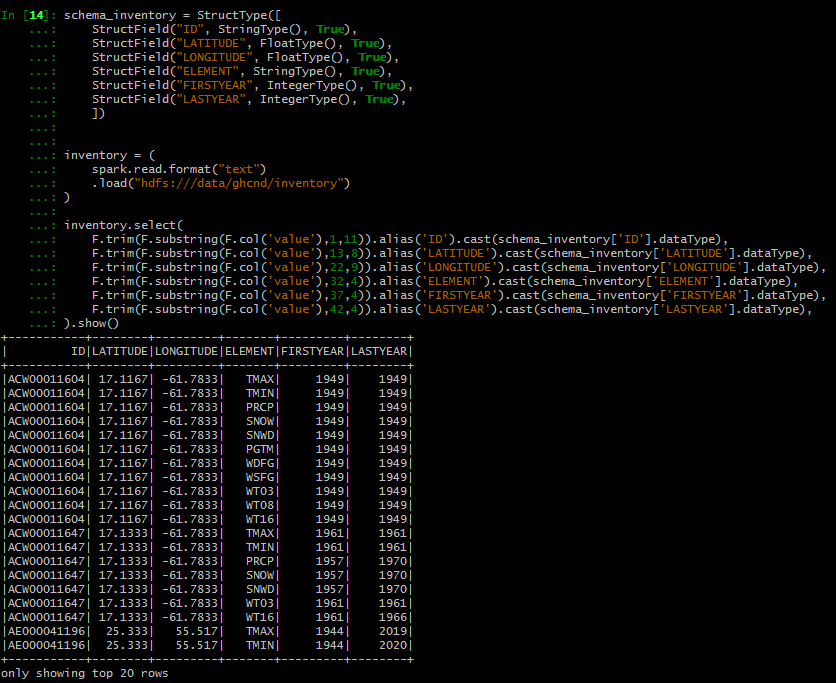
(**c) Load each of stations, states, countries, and inventory into Spark as well. You will need to find a way to parse the fixed width text formatting, as this format is not included in the standard spark.read library. You could try using spark.read.format('text') and pyspark.sql.functions.substring or finding an existing open source library.**

**How many rows are in each metadata table? How many stations do not have a WMO ID?**

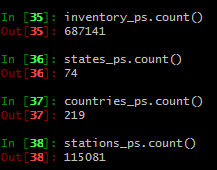




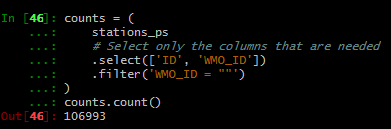




Number of rows in each metadata table is:



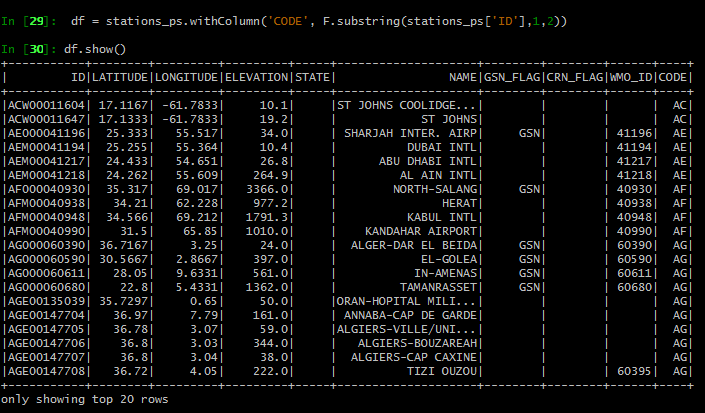
The number of station without a ‘WMO\_ID’



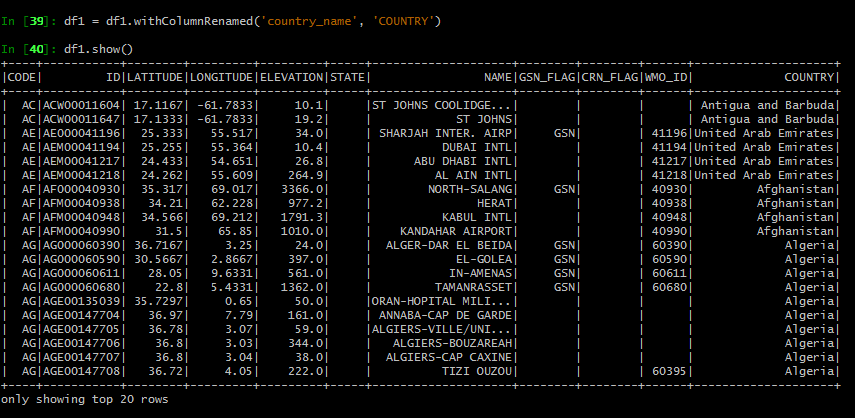
**Q3. Next you will combine the daily climate summaries with the metadata tables, joining on station, state, and country. Note that joining the daily climate summaries and metadata into a single table is not efﬁcient for a database of this size but joining the metadata into a single table is very convenient for ﬁltering and sorting based on attributes at a station level.**

**You will need to start saving some intermediate outputs along the way. Create an output directory in your home directory (e.g. hdfs:///user/abc123/outputs/ghcnd/). Note that we only have 400GB of storage available in total, so think carefully before you write output to HDFS.**

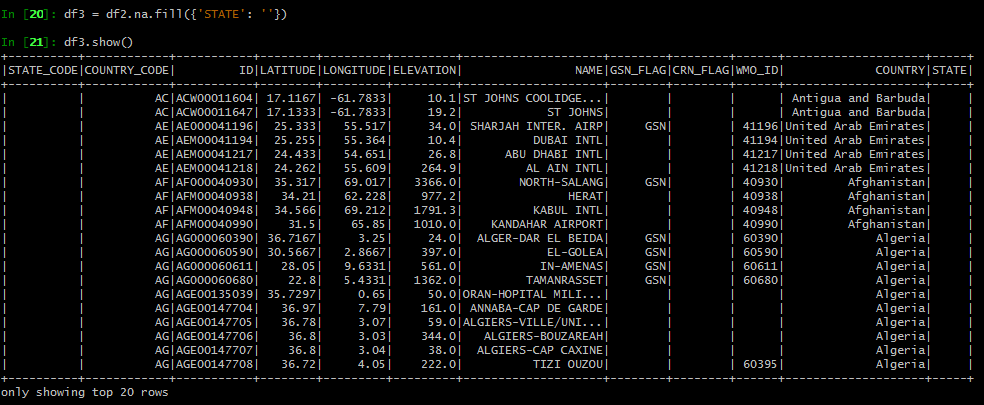
1. **Extract the two-character country code from each station code in stations and store the output as a new column using the withColumn command.**



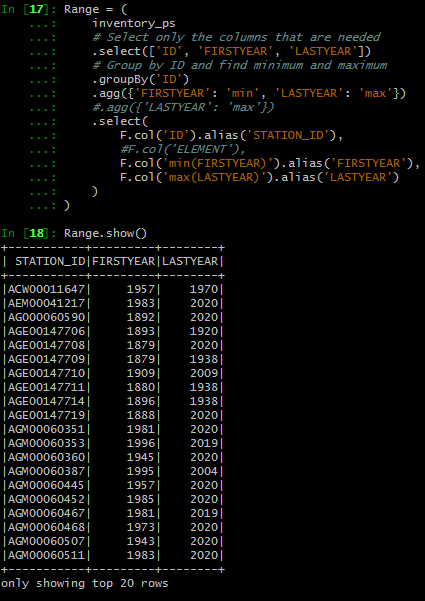
1. **LEFT JOIN stations with countries using your output from part (a).**



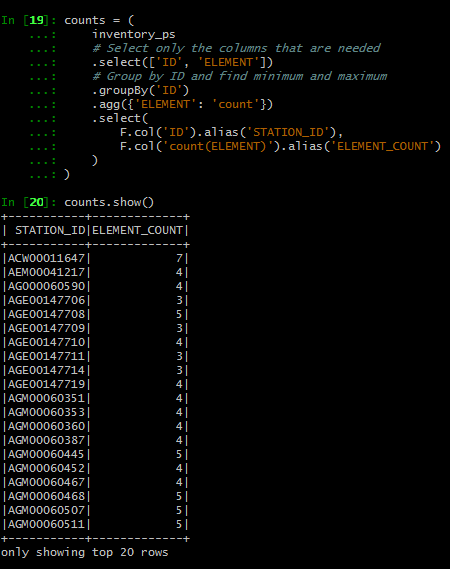
1. **LEFT JOIN stations and states, allowing for the fact that state codes are only provided for stations in the US.**



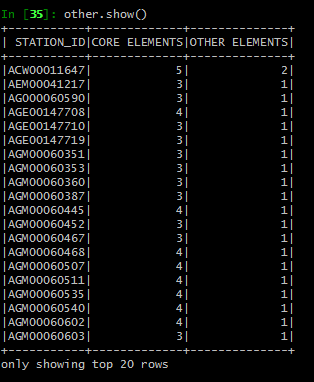
1. **Based on inventory, what was the ﬁrst and last year that each station was active and collected any element at all?**



**How many different elements has each station collected overall?**



**Further, count separately the number of core elements and the number of ”other” elements that each station has collected overall.**

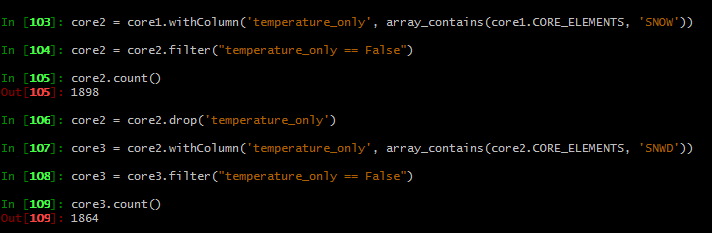


**How many stations collect all ﬁve core elements?**

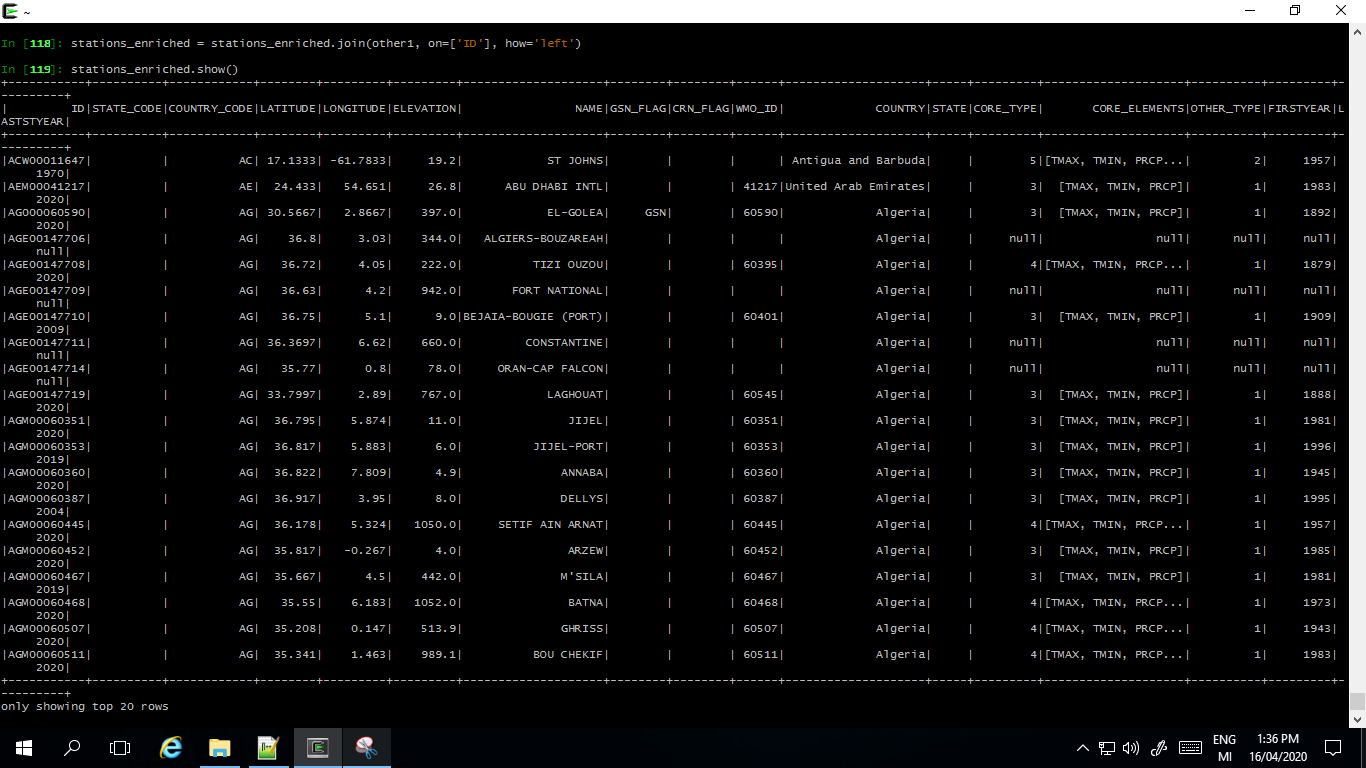
Only one station collects all the 5 core elements.

**How many only collected temperature?**

I have used the array\_contains() method tho check for the other core elements and eliminated those rows, by keeping the rows with temperature elements only. 1864 stations collected only temperature.



1. **LEFT JOIN stations and your output from part (d).**

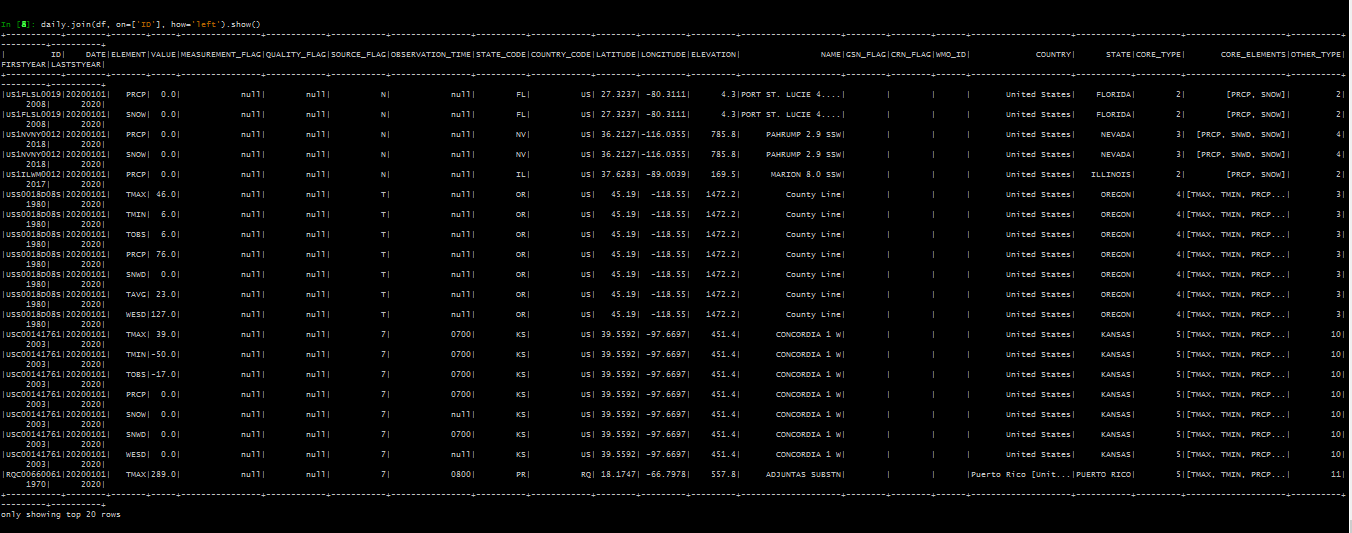


The output is saved in Parquet file format which is more efficient in terms of storage and performance, for Hadoop.

1. **LEFT JOIN your 1000 rows subset of daily and your output from part (e). Are there any stations in your subset of daily that are not in stations at all?**

**How expensive do you think it would be to LEFT JOIN all of daily and stations? Could you determine if there are any stations in daily that are not in stations without using LEFT JOIN?**

Left join daily subset with the stations, returns the below dataframe.



There are no null values in the stationd columns. So it can be assumed that there are no additional station records in the daily. Left Anti Join is used to find any stations in daily that is not in the enriched stationd dataframe, which returned an empty output.



There are only 318 unique stations in the daily subset, that is briefing repeatedly in all these 1000 rows. We do not need all these informations on a daily base. And, also to minimise the complexity it is better to keep it separately.

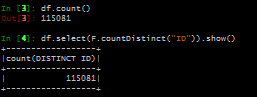
**Analysis**

**Q1. First it will be helpful to know more about the stations in our database, before we study the actual daily climate summaries in more detail.**

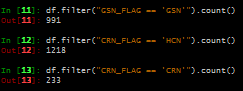
**(a) How many stations are there in total? How many stations have been active in 2020?**

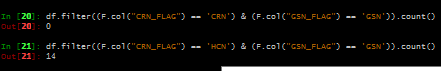
**How many stations are in each of the GCOS Surface Network (GSN), the US Historical Climatology Network (HCN), and the US Climate Reference Network (CRN)? Are there any stations that are in more than one of these networks?**

Altogether there are 115081 unique station IDs in our database. Of these, 28974 have been active in 2020. GCOS Surface Network (GSN) includes 991 stations. 1218 stations in the US Historical Climatology Network(HCN) and 233 in US Climate Reference Network (CRN). 14 stations are in more than one network.



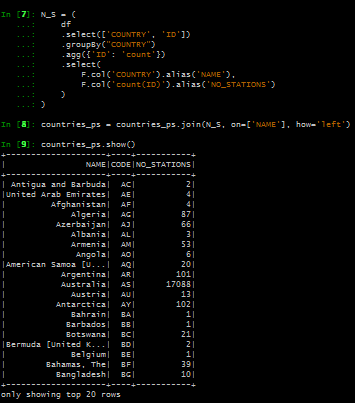


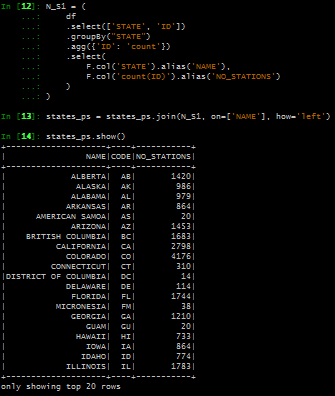




**(b) Count the total number of stations in each country, and store the output in countries using the withColumnRenamed command.**

**Do the same for states and save a copy of each table to your output directory.**



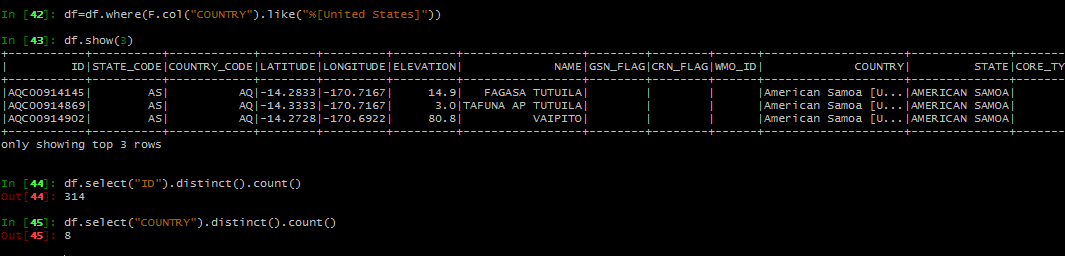


**(c) How many stations are there in the Northern Hemisphere only?**

**Some of the countries in the database are territories of the United States as indicated by the name of the country. How many stations are there in total in the territories of the United States around the world?**

There are 89745 stations in the Northern Hemisphere, and 314 stations in the 8 territories of United States.



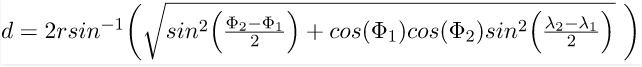


**Q2. You can create user deﬁned functions in Spark by taking native Python functions and wrapping them using pyspark.sql.functions.udf (which can take multiple columns as inputs). You will ﬁnd this functionality extremely useful.**

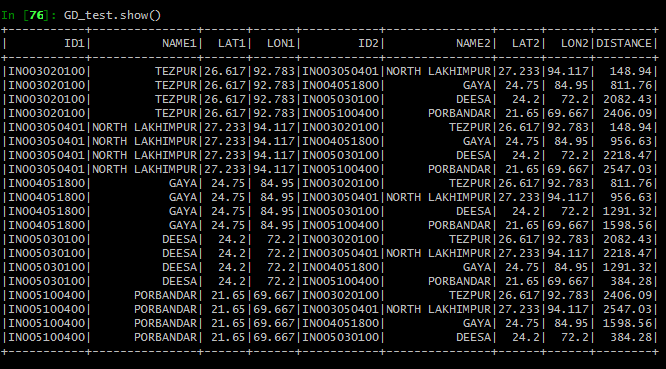
**(a) Write a Spark function that computes the geographical distance between two stations using their latitude and longitude as arguments. You can test this function by using CROSSJOIN on a small subset of stations to generate a table with two stations in each row.**

**Note that there is more than one way to compute geographical distance, choose a method that at least takes into account that the earth is spherical.**

Geometrical distance between the two stations is calculated using Haversine formula.

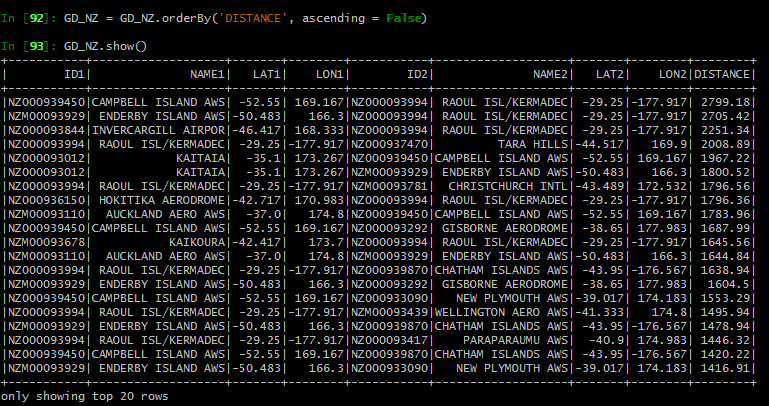


Python function haversine() is converted to a user defined function to operate on a spark dataframe. This udf is applied to the cross joined dataframe to get the desired output.



**(b) Apply this function to compute the pairwise distances between all stations in NewZealand, and save the result to your output directory.**

**What two stations are the geographically furthest apart in New Zealand?**



Duplicate rows are removed using the dropDuplicates() method. The output data frame holds 105 records. Geographically furthest stations in New Zealand is CAMPBELL ISLAND AWS and RAOUL ISL/KERMADEC.

**Q3. Next you will start exploring all of the daily climate summaries in more detail. In order to know how efﬁciently you can load and apply transformations to daily, you need to ﬁrst understand the level of parallelism that you can achieve for the speciﬁc structure and format of daily.**

**(a). Recall the hdfs commands that you used to explore the data in Processing Q1. You would have used**

**hdfs dfs -ls [path] hdfs dfs -du [path]**

**to determine the size of ﬁles under a speciﬁc path in HDFS.**

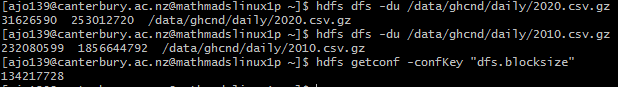
**Use the following command**

**hdfs getconf -confKey "dfs.blocksize"**

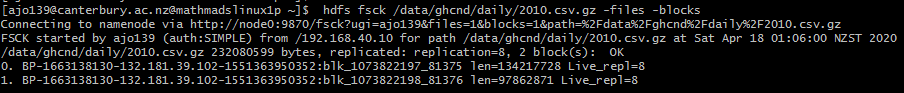
**to determine the default blocksize of HDFS. How many blocks are required for the daily climate summaries for the year 2020? What about the year 2010? What are the individual block sizes for the year 2010? You will need to use another hdfs command.**

**Based on these results, is it possible for Spark to load and apply transformations in parallel for the year 2020? What about the year 2010?**

Files in HDFS are broken into data blocks and stored as individual units. The default size of a data block is 128MB. HDFS data blocks are efficient in handling file sizes, providing fault tolerance and high availability. Moreover, this is a simple concept and the metadata files are stored separately. Parallelism is achieved using the worker nodes.



Daily climate summaries for the year 2020 can be held in one single block. According to the default block size we have for now, 2 blocks were required for the year 2010.



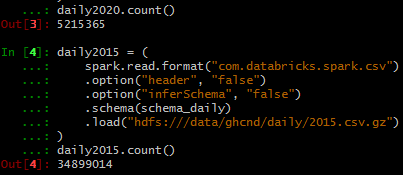
The 2010 file is broken down into 2 data blocks and had the same default block size, 134217728. Both files are located in the same system with same number of data nodes. Spark uses Resilient Distributed Datasets (RDD) to perform parallel processing across these nodes. Irrespective of the number of data blocks Spark can load the data and apply transformations on it, with the same efficiency.

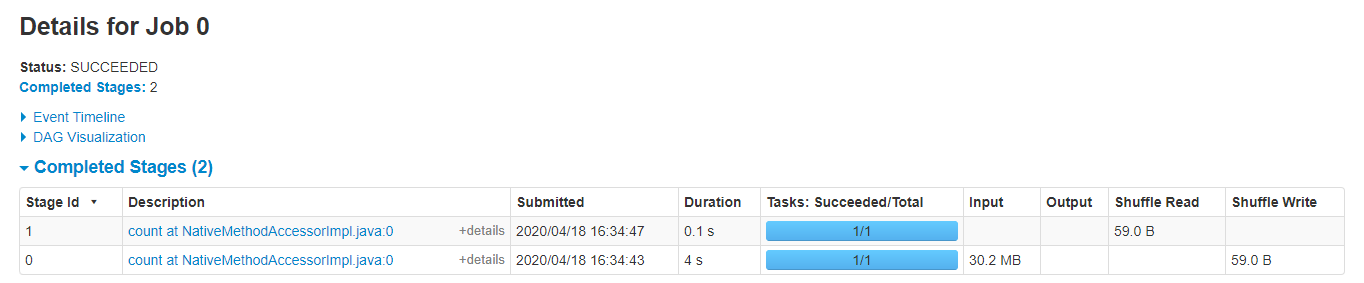
**(b) Load and count the number of observations in daily for each of the years 2015 and 2020.**

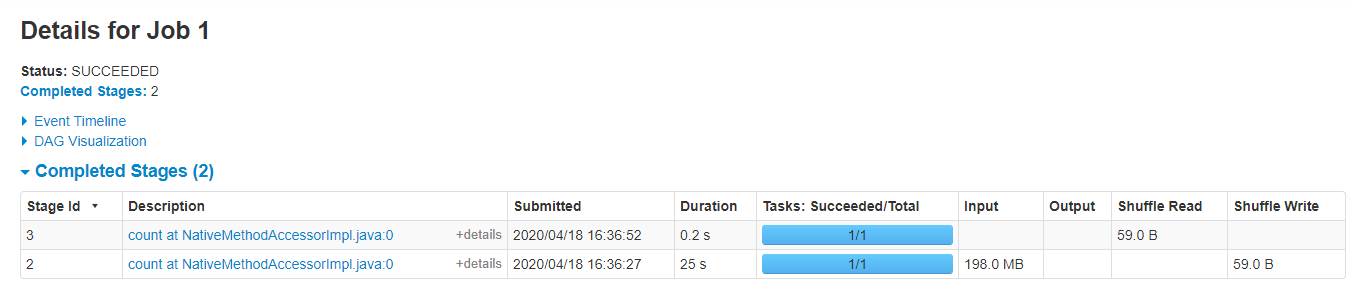
**How many tasks were executed by each stage of each job?**

**You can check this by using your application console. which you can ﬁnd in the web user interface (mathmadslinux1p:8080). You can either determine the number of tasks executed by each stage of each job or determine the total number of tasks completed by each executor after the job has completed.**

**Did the number of tasks executed correspond to the number of blocks in each input?**







There were 2 stages for each of the 2 jobs. The first stage executed writing in 1 task and the second stage executed reading in 1 another task tasks.

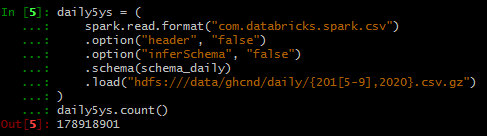
The storage and the processing are the 2 separate core components on Hadoop. Thus the number of tasks executed is not corresponding to the number of input blocks.

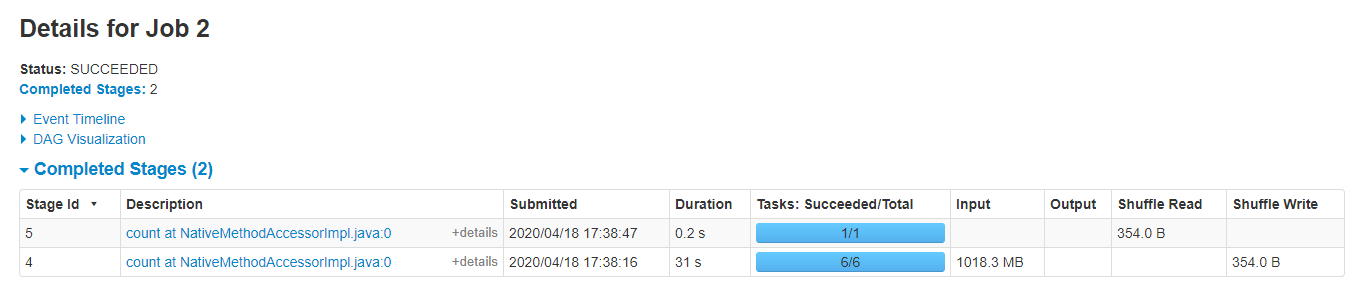
**(c) Load and count the number of observations in daily from 2015 to 2020.**

**Note that you can use any regular expressions in the path argument of the read command.**

**Now how many tasks were executed by each stage, and how does this number correspond to your input?**

**Find out how Spark partitions input ﬁles that are compressed.**





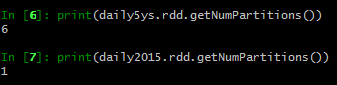
The initial stage executed 6 tasks and the second stage executed 1 task for this job. The input size for this job is 1018MB, that demanded more resources and transformation tasks to achieve parallel processing. The job is done with 4 executors together. The output is generated in 31 seconds just 5 seconds more than the previous job that had an input size of just 198MB. More executors on the cluster is important to increase parallelism.

Compressed files reduce the file sizes and are helpful in handling data transfers across the nodes. When it comes to distributed computing partitioning of these files is a challenge, when the format is not splittable. Apache spark uses the codecs provided by Hadoop to deal with these files. Splittable compressed files are processed as any other file. Not splittable compressed files are processed by a single executor. (waitingforcode.com)

**(d) Based on parts (b) and (c), what level of parallelism can you achieve when loading and applying transformations to daily? Can you think of any way you could increase this level of parallelism either in Spark or by additional preprocessing?**

The daily data set is 15.5 GB. Improved level of parallelism can be obtained by more executors and more resources. Another method is by way of managing spark partitions.

For large volume of data processing in spark, data partitioning is critical. An atomic chunk of data stored on a node in the cluster is called a partition in spark and are basic units of parallelism. Apache Spark manages data through RDDs. By default a partition is created for every [HDFS](https://www.dezyre.com/hadoop-course/hdfs)partition of size 64MB. Apache Spark can run a single concurrent task for every partition of an RDD, up to the total number of cores in the cluster. The best way to decide on the number of partitions in an RDD is to make the number of partitions equal to the number of cores in the cluster so that all the partitions will process in parallel and the resources will be utilized in an optimal way. (ProjectPro) We need to understand how data is partitioned and repartition method can be used to modify it.

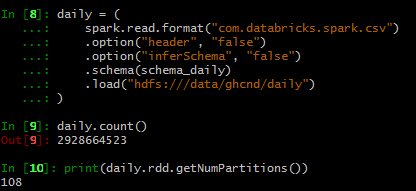


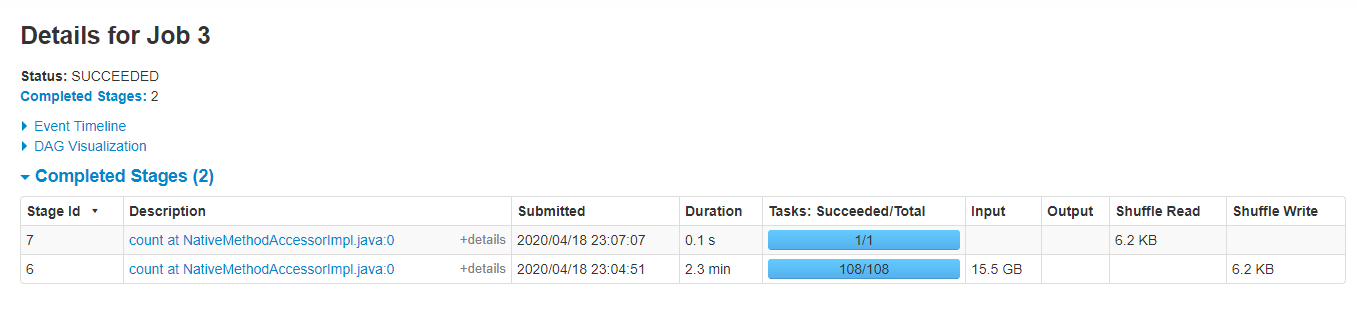
**Q4 All of the data stored in HDFS under hdfs:///data/ghcnd has a replication factor of 8 and is available locally on every node in our cluster. As such you will always be able to load and apply transformations to multiple years of daily in parallel.**

**Again,youmaywanttouseonlypartofthedailyclimatesummarieswhileyouarestilldeveloping your code so that you can use fewer resources to get preliminary results without waiting all day.**

**(a) Count the number of rows in daily.**

**Note that this will take a while if you are only using 2 executors and 1 core per executor, and that the amount of driver and executor memory should not matter unless you actually try to cache or collect all of daily. You should never try to cache or collect all of daily.**



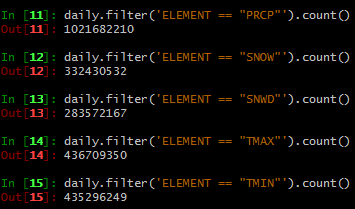


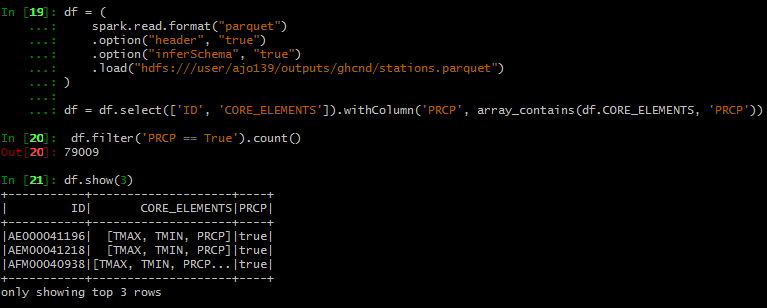
15.5 GB of data has got 108 partitions. Apparently, number of partitions is same as the number of tasks performed in stage 1. This is perfectly in concordance with the number of cores, where each of the 8 cores have to work on 12 partitions each. The job is executed in 2 stages and 109 tasks in 2.4 minutes. This is the level of parallelism we achieved using 4 executors, 8 cores and 4GB each of the worker and master memories. If we double the resources, the task will be accomplished half the time.

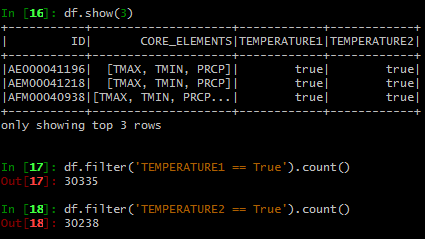
**(b) Filter daily using the filter command to obtain the subset of observations containing the ﬁve core elements described in inventory.**

**How many observations are there for each of the ﬁve core elements?**

**Which element has the most observations? Is this result consistent with Processing Q3?**







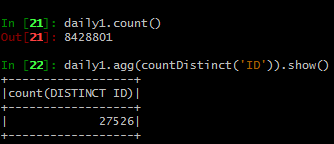
The precipitation element PRCP has the most observations. Out of the 115081 unique stations in the data 79009 stations were collecting PRCP from Q3 part of Processing. Only 30335 & 30238 stations were collecting data on TMAX and TMIN respectively.

**(c) Many stations collect TMAX and TMIN, but do not necessarily report them simultaneously due to issues with data collection or coverage. Determine how many observations of TMIN do not have a corresponding observation of TMAX.**

**How many different stations contributed to these observations?**

I filtered out the other elements, used the groupBy() function on ID and DATE column to produce the distinct combinations of these columns, and then used the pivot() function to pivote the ELEMENT column with its VALUE. The output is again filtered for the TMAX column with null values where TMIN is not a null value.





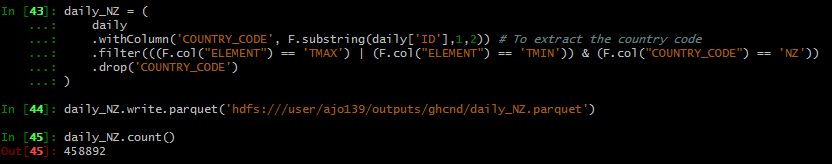
8428801 observations of TMIN do not have a corresponding observation of TMAX and 27526 stations contributed to these observations.

**(d) Filter daily to obtain all observations of TMIN and TMAX for all stations in New Zealand and save the result to your output directory.**

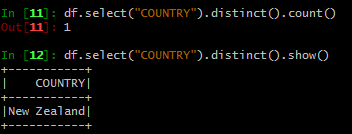
**How many observations are there, and how many years are covered by the observations?**

**Use hdfs dfs -copyToLocal to copy the output from HDFS to your local home directory and count the number of rows in the part ﬁles using the wc -l bash command. This should match the number of observations that you counted using Spark.**

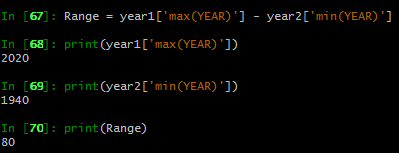
**Plot time series for TMIN and TMAX on the same axis for each station in New Zealand using Python, R, or any other programming language you know well. Also, plot the average time series for TMIN and TMAX for the entire country.**



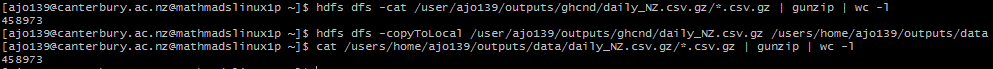
I have used distinc() on column COUNTRY name to see if there any territories included.







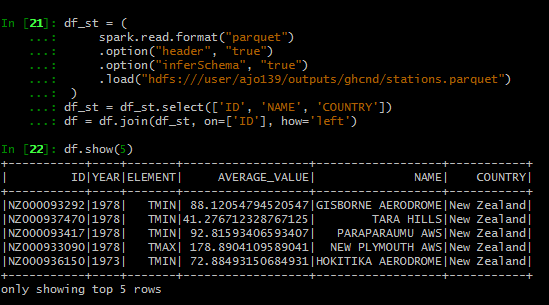
There are 458892 observations in the daily New Zealand file. The observations are covered from the year 1940 to 2020 (80years).



I could not count the rows from my parquet file, that gave me count of just 1-part file (4362) even after trying all the ways I could. I tried minimizing the number of partitions using coalesce(1) function. I even tried installing the parquet tools but failed. Finally, I changed the format to .csc.gz.

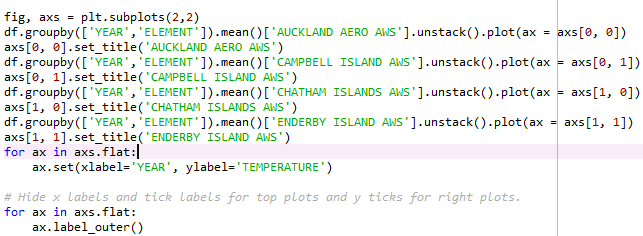
However, when I checked the number of rows using wc -l bash command, the result is slightly different 458973. This add on is due to the extra headers in each part file.

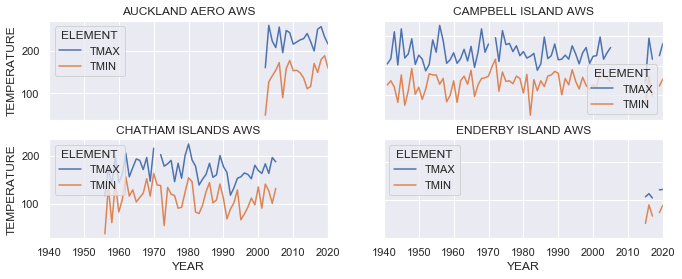
To plot the time series, I have taken the yearly average of values, and joined with the stations to get the station names.

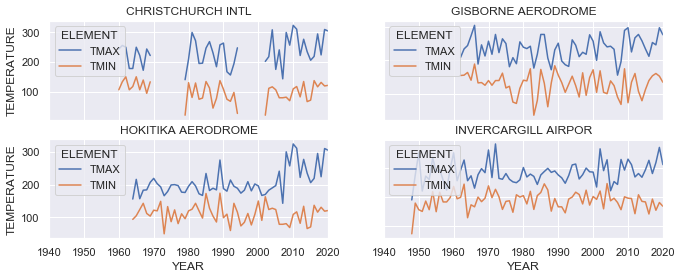


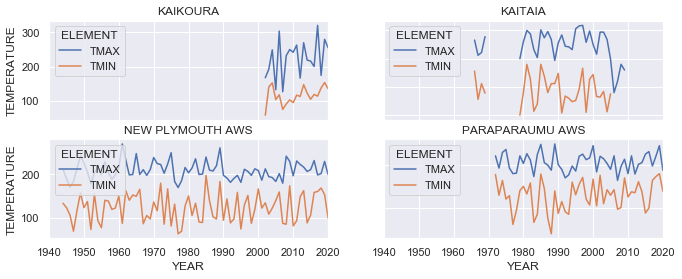


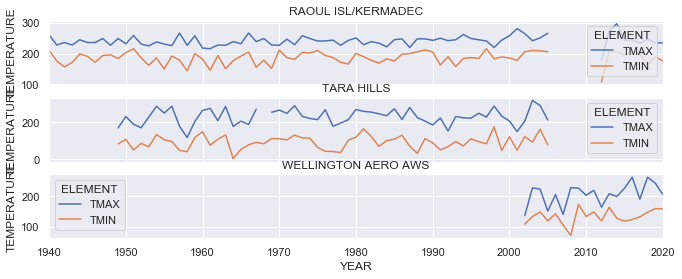
Time series temperature for each station in New Zealand. I used pandas and matplotlib from python, for data visualization.

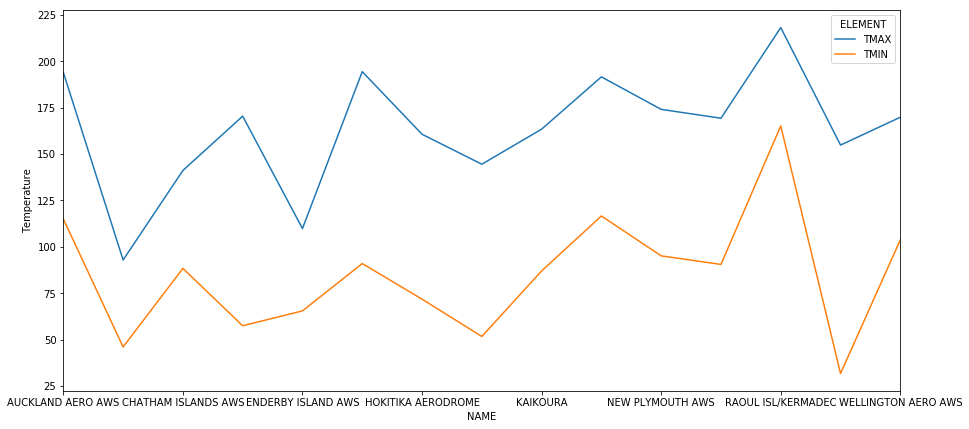




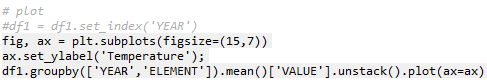


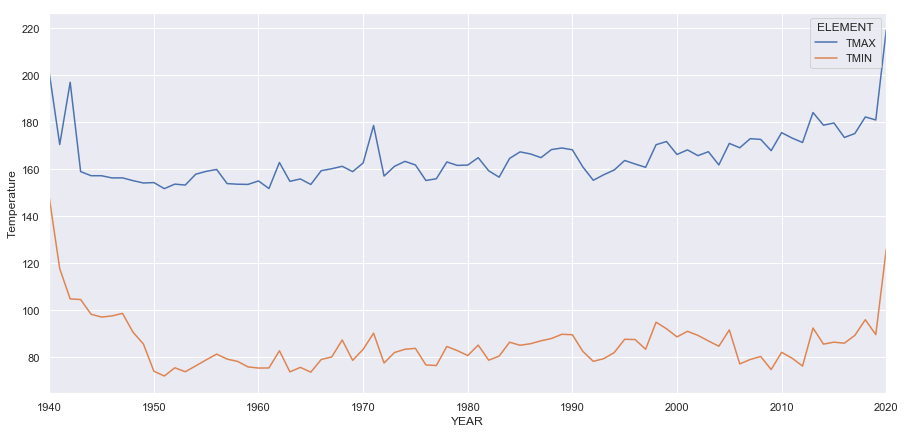






Time series temperature for the entire New Zealand



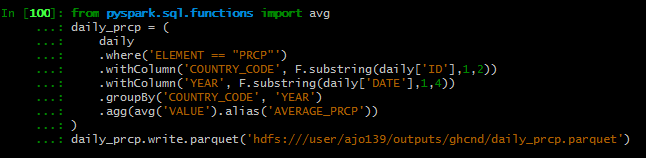


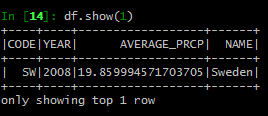
**(e) Group the precipitation observations by year and country. Compute the average rainfall in each year for each country and save this result to your output directory.**

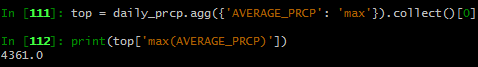
**Which country has the highest average rainfall in a single year across the entire dataset? Is this result sensible? Is this result consistent?**

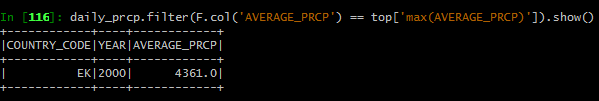
**Find an elegant way to plot the cumulative rainfall for each country using Python, R, or any other programming language you know well. There are many ways to do this in Python and R speciﬁcally, such as using a choropleth to color a map according to average rainfall.**

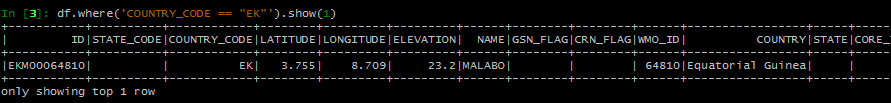
Average rainfall groupby year and countrycode is obtained from daily. This file is joined with the station data to obtain the country name.



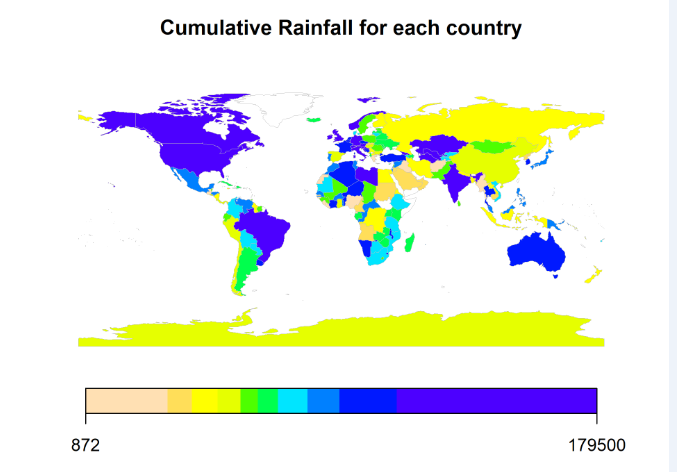








Equatorial Guinea had the highest average rainfall in the year 2000, across the entire data set. Among the top 100 list this country appeared 4 times (1996,1997,2000 & 2001). But average precipitation values show high variations (4361,1100,709 & 576). These variations are suspectable.



I tried to plot cumulative rainfall using different file formats and libraries. But I was kept on getting the errors in one way or another. For some reason I could not get it displayed. I have submitted the code that I worked in R. Since the file size for the geospatial shapefiles were very huge, I had to find out a way to subsetting the data. But all attempt failed.

